**Practica 4:**

**Construcción de árboles de sintaxis abstracta**

**Grupo 11:**

**Youssef El Faqir El Rhazoui**

**Enrique Ávila Rodríguez**

1. **Conjunto de funciones constructoras**

**Prog:** LDs **x** LIs 🡪 Prog

**LD\_simp:** String **x** String 🡪 D

**LD\_comp:** String **x** String **x** LDs 🡪 LDs

**LI\_simp:** String **x** Exp 🡪 I

**LI\_comp:** String **x** Exp **x** LIs 🡪 LIs

**Mas:** Exp **x** Exp 🡪 Exp

**Menos:** Exp **x** Exp 🡪 Exp

**And:** Exp **x** Exp 🡪 Exp

**Or:** Exp **x** Exp 🡪 Exp

**Distinto:** Exp **x** Exp 🡪 Exp

**Igual:** Exp **x** Exp 🡪 Exp

**Menor\_que:** Exp **x** Exp 🡪 Exp

**Menor\_igual\_que:** Exp **x** Exp 🡪 Exp

**Mayor\_que:** Exp **x** Exp 🡪 Exp

**Mayor\_igual\_que:** Exp **x** Exp 🡪 Exp

**Mul:** Exp **x** Exp 🡪 Exp

**Div:** Exp **x** Exp 🡪 Exp

**Not:** Exp 🡪 Exp

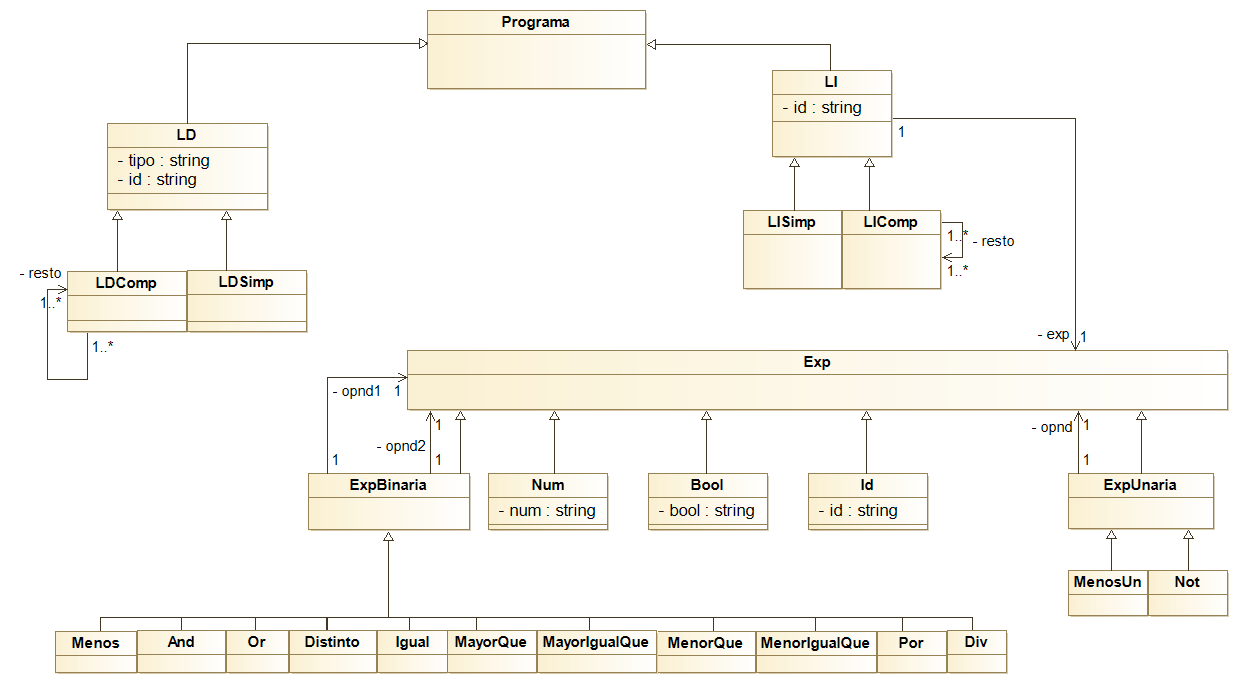
**Menos\_unario:** Exp 🡪 Exp

**Num:** String 🡪 Exp

**Bool:** String 🡪 Exp

**Id:** String 🡪 Exp

1. **Diagrama de clases**



1. **Gramática de atributos**

Prog 🡪 LDs && LIs

Prog.a = prog(LDs.a, LIs.a)

LDs 🡪 LDs; D

.a = ldCompuesta(D.tipo, D.id, .a)

LDs 🡪 D

LDs.a = ldSimple(D.tipo, D.id)

LIs 🡪 LIs; I

.a = liCompuesta(I.id, I.exp, .a)

LIs 🡪 I

LIs.a = liSimple(I.id, I.exp)

D 🡪 tipo identificador

D.tipo = tipo.lex

D.id = identificador.lex

I 🡪 identificador = Exp0

I.id = identificador.lex

I.exp = Exp0.a

Exp0 🡪 Exp0 Op0 Exp1

.a = mkexp(Op0.op, .a, Exp1.a)

Exp0 🡪 Exp1

Exp0.a = Exp1.a

Exp1 🡪 Exp2 and Exp1

.a = and(Exp2.a, .a)

Exp1 🡪 Exp2 or Exp2

Exp1.a = or(.a, .a)

Exp1 🡪 Exp2

Exp1.a = Exp2.a

Exp2 🡪 Exp3 Op2 Exp3

Exp2.a = mkexp(Op2.op, .a, .a)

Exp2 🡪 Exp3

Exp2.a = Exp3.a

Exp3 🡪 Exp3 Op3 Exp4

.a = mkexp(Op3.op, .a, Exp4.a)

Exp3 🡪 Exp4

Exp3.a = Exp4.a

Exp4 🡪 - Exp4

.a = menos\_unario(.a)

Exp4 🡪 not Exp5

Exp4.a = not(Exp5.a)

Exp4 🡪 Exp5

Exp4.a = Exp5.a

Exp5 🡪 numero

Exp5.a = num(numero.lex)

Exp5 🡪 booleano

Exp5.a = bool(booleano.lex)

Exp5 🡪 identificador

Exp5.a = id(identificador.lex)

Exp5 🡪 (Exp0)

Exp5.a = Exp0.a

Op0 🡪 +

Op0.op = “+”

Op0 🡪 -

Op0.op = “-”

Op2 🡪 !=

Op2.op = “!=”

Op2 🡪 ==

Op2.op = “==”

Op2 🡪 <

Op2.op = “<”

Op2 🡪 <=

Op2.op = “<=”

Op2 🡪 >

Op2.op = “>”

Op2 🡪 >=

Op2.op = “>=”

Op3 🡪 \*

Op3.op = “\*”

Op3 🡪 /

Op3.op = “/”

**Definimos la función mkexp como sigue:**

fun mkexp(op, opnd1,opnd2) {

switch(op) {

"+" => return suma(opnd1,opnd2)

"-" => return resta(opnd1,opnd2)

"!=" => return distinto(opnd1,opnd2)

"==" => return igual(opnd1,opnd2)

"<" => return menorQue(opnd1,opnd2)

"<=" => return menorIgualQue(opnd1,opnd2)

">" => return mayorQue(opnd1,opnd2)

">=" => return mayorIgualQue(opnd1,opnd2)

"\*" => return mul(opnd1,opnd2)

"/" => return div(opnd1,opnd2)

}

}

1. **Acondicionamiento para imp descendente**

Prog 🡪 Sec\_Dec && LIs

Prog.a = prog(LDs.a, LIs.a)

LDs 🡪 D PDec

PDec.ah = ldSimple(D.tipo, D.id)

LDs.a = PDec.a

PDec 🡪 ; D PDec

.a = ldCompuesta(.ah, D.a)

.a = .a

PDec 🡪 ε

PDec.a = PDec.ah

LIs 🡪 I PIns

PIns.ah = liSimple(I.id, I.exp)

LIs.a = PIns.a

PIns 🡪 ; I PIns

.a = liCompuesta(.ah, I.a)

.a = .a

PIns 🡪 ε

PIns.a = PIns.ah

D 🡪 tipo identificador

D.tipo = tipo.lex

D.id = identificador.lex

I 🡪 identificador = Exp0

I.id = identificador.lex

I.exp = Exp0.a

Exp0 🡪 Exp1 RExp0

RExp0.ah = Exp1.v

Exp0.v = RExp0.v

RExp0 🡪 Op0 Exp1 RExp0

.ah = mkexp(Op0.op, .ah, Exp1.v)

.a = .a

RExp0 🡪 ε

RExp0.a = RExp0.ah

Exp1 🡪 Exp2 RExp1

RExp1.ah = Exp2.a

Exp1.a = RExp1.a

RExp1 🡪 and Exp1

RExp1.a = and(RExp.ah, Exp1.a)

RExp1 🡪 or Exp2

RExp1.a = or(RExp.ah, Exp2.a)

RExp1 🡪 ε

RExp1.a = RExp1.ah

Exp2 🡪 Exp3 RExp2

RExp2.ah = Exp3.a

Exp2.a = RExp2.a

RExp2 🡪 Op2 Exp3

RExp2.a = mkexp(Op2.op, RExp2.ah, Exp3.a)

RExp2 🡪 ε

RExp2.a = RExp2.ah

Exp3 🡪 Exp4 RExp3

RExp3.ah = Exp4.a

Exp3.a = RExp3.a

RExp3 🡪 Op3 Exp4 RExp3

.ah = mkexp(Op3.op, .ah, Exp4.a)

.a = .a

RExp3 🡪 ε

RExp3.a = RExp3.ah

Exp4 🡪 - Exp4

.a = menos\_unario(.a)

Exp4 🡪 not Exp5

Exp4.a = not(Exp5.a)

Exp4 🡪 Exp5

Exp4.a = Exp5.a

Exp5 🡪 numero

Exp5.a = num(numero.lex)

Exp5 🡪 booleano

Exp5.a = bool(booleano.lex)

Exp5 🡪 identificador

Exp5.a = id(identificador.lex)

Exp5 🡪 (Exp0)

Exp5.a = Exp0.a

Op0 🡪 +

Op0.op = “+”

Op0 🡪 -

Op0.op = “-”

Op2 🡪 !=

Op2.op = “!=”

Op2 🡪 ==

Op2.op = “==”

Op2 🡪 <

Op2.op = “<”

Op2 🡪 <=

Op2.op = “<=”

Op2 🡪 >

Op2.op = “>”

Op2 🡪 >=

Op2.op = “>=”

Op3 🡪 \*

Op3.op = “\*”

Op3 🡪 /

Op3.op = “/”